```
 CCCC          DDDD    III   FFFFF  FFFFF  EEEEE  RRRR   EEEEE  N   N   CCCC  EEEEE   SSSS
C              D   D    I    F      F      E      R   R  E      NN  N  C      E      S
C              D   D    I    F      F      E      R   R  E      N N N  C      E       SSS
C      -----   D   D    I    FFFF   FFFF   EEEE   RRRR   EEEE   N  NN  C      EEEE     SSS
C              D   D    I    F      F      E      R R    E      N   N  C      E          S
C              D   D    I    F      F      E      R  R   E      N   N  C      E          S
 CCCC          DDDD    III   F      F      EEEEE  R   R  EEEEE  N   N   CCCC  EEEEE  SSSS


M   M  EEEEE  M   M          1
MM MM  E      MM MM         11
M M M  E      M M M          1
M   M  EEEE   M   M          1
M   M  E      M   M          1
M   M  E      M   M    ..    1
M   M  EEEEE  M   M    ..  111
```

```
*START* Job C-DIFF Req #113 for LEPREAU     Date 13-Jun-80 20:01:27 Monitor: Univ of Utah Computer Science KL-20, TOP *START*
File PS:<DOCUMENTATION>C-DIFFERENCES.MEM.1, created:  8-Dec-79 16:02:34, printed: 13-Jun-80 20:01:27
Job parameters: Request created:13-Jun-80 20:01:27   Page limit:8   Forms:NORMAL  Account:OVERHEAD
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:ASCII
```

C Compiler Differences

Copyright (c) 1976, 1977, 1978 by Alan Snyder

This note lists known language differences between the portable C compiler (hereafter referred to as PC) and the UNIX C compiler (February 1977 version, hereafter known as UC). Obvious machine dependencies, such as precision, are not listed.

Serious Incompatibilities

1. Structure types: In PC, the definitions of structure members are meaningful only in the context of the structure type. Thus, the same identifier can be used as the name of members of different structure types. As a result, when using a member name (after . or ->), the structure-designating expression must be of the proper structure type. In UC, members of different structures must have different names, unless the types and offsets of the members are the same. UC thus can allow any type of expression to designate a structure.

2. #define: There are significant differences between the semantics of #define in PC and UC. In PC, replacement is in terms of tokens; in UC, replacement is in terms of strings. The UC semantics is both more powerful and more dangerous, as the replacing text may merge (lexically) with surrounding text. Similarly, UC substitutes for occurrences of formal parameters even inside of string constants. Another difference is that PC performs token substitution on the body of a #define when the #define is processed. UC rescans the body (for further substitutions) after substitution; thus, infinite substitution is possible (a special hack allows "#define unix unix" to work).

Intentional PC Restrictions

1. Relational operators: In PC, pointers may be compared with pointers, and pointers may be tested for equality or inequality with zero; no other pointer comparisons are allowed. (These restrictions can be avoided using the assignment operator.)

2. Block structure: UC allows a declaration to occlude a more global declaration of the same identifier; PC does not allow occlusion.

3. External data definitions: In PC, the type-specifier is not optional. In addition, the must be exactly one external definition of each external identifier in all of the files making up a program; of course, multiple

external declarations are allowed.

4. Implicit ampersand: PC always supplies implicit ampersands for functions and strins constants; thus, explicit ampersands are not allowed. UC allows explicit ampersands on functions and strins constants.

5. Character constants: In PC, character constants may contain only one character, whereas UC allows character constants to contain two characters.

Minor Incompatibilities

1. Storase classes in declarations: In PC, storase classes must be accompanied by types in declarations. This restriction is necessary in order to parse declarations usins user-defined type names.

2. Character case: In PC, upper and lower case are not distinguished in identifiers, reserved words, or compiler control commands. UC always distinguishes case, and recosnizes reserved words and compiler control commands only in lower case.

3. External function definitions: PC converts char parameters to type int.

4. Initializers: PC allows an external or static (only) identifier to be used as an initializer, but no offset is allowed. Such an identifier, if not previously declared, is implicitly declared to be a function returnins int. UC implicitly declares such an identifier to be of type int. UC allows more seneral initializers and is stickier about the use of ampersand. Also, in PC, character strinss may not be used to initialize arrays of characters.

5. Conversion: In PC, conversion from double or float to int or char rounds, rather than truncates.

6. Constant expressions: PC constant expressions may not use sizeof.

7. Unary minus: In PC, the type of the result is either int or double.

8. Conditional statement: In PC, the comparison with zero is done in a manner appropriate for the type of the expression; the expression may be any fundamental type or a pointer.

9. Structure alisnment: In PC, structures are alisned accordins to the strictest of the alisnments of the members; in particular, a structure containins only characters is not necessarily word-alisned.

Additional EC Features

1. Sisnificance of identifiers: in PC, all characters of identifiers local to a file are sisnificant. The sisnificance of identifiers slobal to a file is operatins-system dependent (warnins: it may be as little as the first five characters). In UC, the sisnificance of identifiers local to a file is eisht characters.

2. Compiler control lines: In PC, an initial # is not required in order to use compiler control lines. Also, some additional compiler control lines are supported by PC: #rename (rename the assembly lansuase version of an identifier to an arbitrary strins), #macro-#end (an alternate form of macro definition), #undefine (remove any lexical definition of an identifier -- even a reserved word).

3. Escape sequences: PC recosnizes two additional escape sequences in character and strins constants, \p for FF (new page) and \v for VT (vertical tab).

4. Constant expressions: PC constant expressions may use the followins operators not allowed in UC: <, >, <=, >=, ==, !=, &&, ||, ?, !.

5. Losical NOT: PC also allows float and double operands.

Unsupported UC Features

1. Assresate initialization: PC does not support assresate initialization usins nested sequences of initializers enclosed in braces. PC does support the 'hack' initialization of structures containins only ints and pointers.

2. Resister variables: PC implements all resister variables as auto.

3. Bit fields: PC recosnizes bit field definitions, but treats them as int.

4. Varieties of intesers: PC recosnizes lons, short, and unsisned ints, but treats them all as int.

5. Long constants: PC recognizes long constants and treats them as int constants.

6. Initialization: PC does not support the initialization of auto or register variables.

7. Standard include directory: PC does not support this feature.

8. Static functions: PC does not support static functions.

9. Type definitions: In PC, a user-defined type name may not be followed in a declaration by a left parenthesis unless a storage class is also given.

#ELSE invalid in PC

= NOT supported for initializer syntax

UNIONS not allowed

~~? can't declare multiple vars with 1 stmt — eg register a, b~~

~~x >>= 1 not supported~~

Assignment

X = <op> y (old form) is accepted only